Reference Guide

# opendelight

A PHP Application Development Framework

Ashwini Rath

BATOI PRESS

# Reference Guide

**Last Update: Friday, 06 September 2013**



**BATOI PRESS**
***press.batoi.com***
421, Saheed Nagar, Bhubaneswar 751 007 (INDIA)
Publisher Website: https://www.batoi.com/press
Email: press@batoi.com

# Table of Contents

# Get Started with Opendelight

Opendelight is easy to use for your PHP application development and has an insignificant or no learning curve. On this page, you will find a few easy steps to start using the Opendelight framework.

First, you need to make sure that you have a standard PHP environment with PDO (PHP Data Object) enabled. If you are new to PDO, please refer to its official web pages at http://php.net/manual/en/book.pdo.php. You can choose any database that is supported by PDO like MySQL, SQLite, Oracle, PGSQL, etc.

Download the compressed file (choose appropriate format as suitable for your platform) from Download page. Read information provided on Download page for any further clarifications.

After you download, please follow instructions on Installation Page to install the framework – a task of a few minutes : )

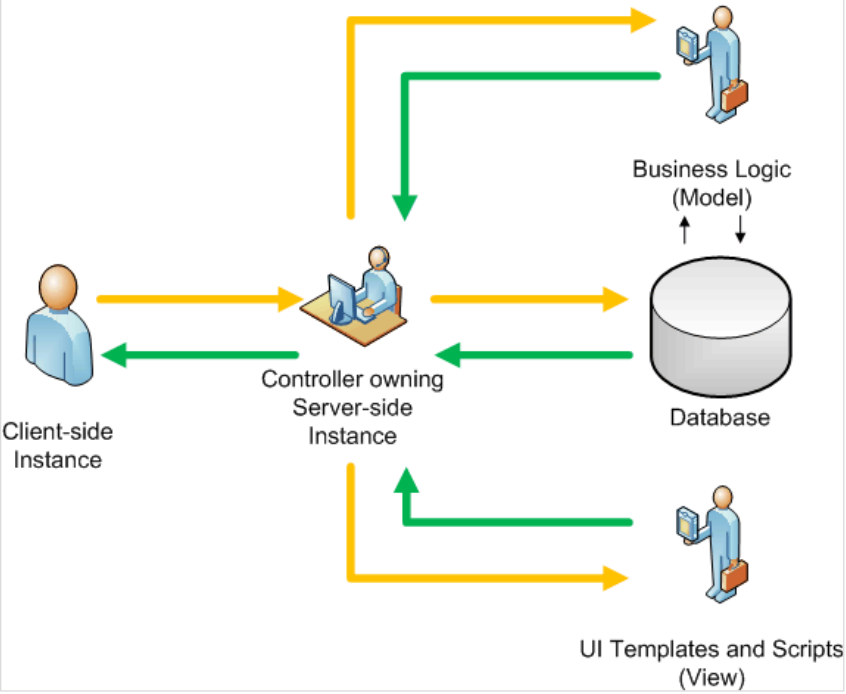
Once you install, you are done with your application setup, security procedures, access privilege settings and signin forms. Now, signin at your Delight IDE to start developing your application. You can find all requisite information on the comprehensive documentation available online to read and download.

Happy developing application!

# Overview of Framework Architecture

The Opendelight framework encompasses multi-tier architecture of web application, and is based on several design patterns including most notably Model-View-Controller (MVC). Currently, opendelight uses jQuery javascript framework and jQuery UI CSS framework with appropriate extensions for achieving UI rendering, effects, asset management, user interactions and functions at client-side (however, you are free to use any other client-side framework).

An instance of application at client-side invokes instance at server-side through HTTP request which is responsible for processing business logic of the application, and in turn, provides data back to client-side instance for further actions (rendering, effects, asset management, user interactions and functions).

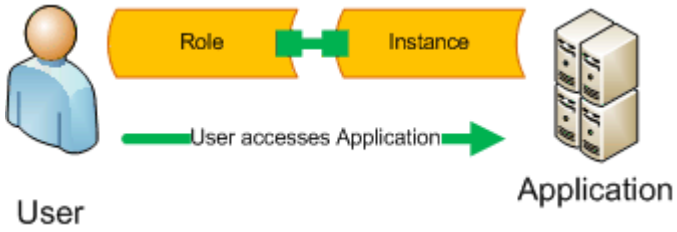



**(Illustration of Opendelight Framework Architecture)**

Every HTTP request is received by a Controller at the server-side, and each HTTP request is defined uniquely by the Event ID (through $_REQUEST[ID] parameter) that is passed from the client-side application instance. Based on Event ID, the

Controller validates the HTTP request and the user sending the request.

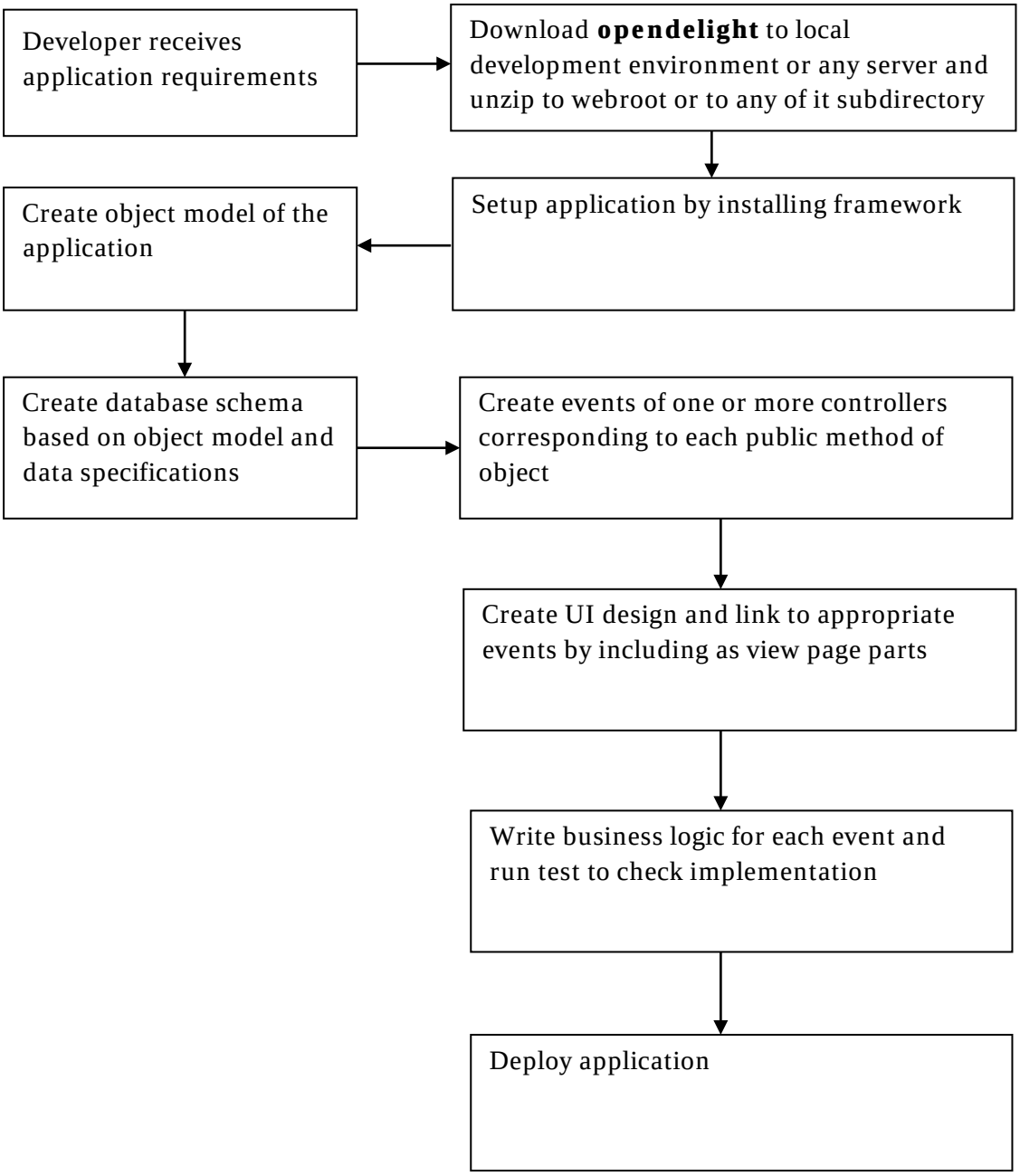
The framework provided an in-built access control scheme based on user roles, called Role Based Access Control (RBAC) scheme, that allows access to application by its users depending on their roles assigned to. The basic concept of the scheme lies in treating application and user accessing the application as two objects, and basing their interaction through user's role associated with application instance.




**(Illustration of Role-Based Access Control Scheme)**

Upon successful validation, parameters collected from HTTP request are passed to BL objects to execute the request. Model (consisting of classes and independent scripts – to give life to PHP old-timers : ) ) carries out BL processing, and returns resulting data in the form of arrays. After successful interaction with Model, Controller passes the available resulting data to View for composing the UI, if required. View creates the final client-side code (all in XHTML, JavaScript, CSS, etc.) and then sends the resulting code (or data) to Controller. Finally, Controller sends the final data to client-side application instance completing the server-side process of data-flow.

# Typical Developer Workflow



# Glossary of Terms Used in Framework

Please find the list of terms that are used in Opendelight framework and their description below:

## Admin

The username `admin` (case-sensitive) is the super user which can access Opendelight IDE and can develop, manage and access entire application. This user gets such a right through a default role assigned to it – `Administrator`. Neither this username nor its association with the role `Administrator` can be modified.

## Administrator Role

When you install the framework, the default role is `Administrator`, and it is assigned automatically to the user `admin`. Neither this role nor its association with the user `admin` can be modified.

## Application Class

An Application Class refers to any PHP class that contains business logic of the application and is a part of **Model**. Application classes can be created and managed directly through IDE. However, you can also edit these classes in your favourite code editor too. These classes are located in `/model/class` directory. For more details on Application Class, please go to https://www.batoi.com/opendelight/docs/model

## Application Settings

Application settings refer to setup information that are used while installing the framework, thus setting up the application with Opendelight framework. This includes *application name*, *description*, *author name*, *email address of admin user*, *URL of application* and *database settings*. Database settings include *DSN*, *Database Name*, *DB user* and *Password*, and *Table Prefix* (the string that will be preppended to each table of the application including **Opendelight Database Tables**). For more details on Application Settings, please go to https://www.batoi.com/opendelight/docs/ide

## Configuration Variables

While developing an application, we tend to use a number of

magic numbers. Ideally we store these in a separate file, and in turn, include it everytime within the application. With Opendelight, we do not need to do any such effort; rather we create these as configuration variables on IDE, and then use them as a property (date type is associative array) of `$APP` object, `$APP->CONFIG`, within the application. For more details on Configuration Variables and how to manage these, please go to https://www.batoi.com/opendelight/docs/model

## Controller

A Controller is a PHP file that takes the HTTP request, and manages the control of the execution path of the application (instance of application) at server-side. For more details on Controller and how to use them, please go to https://www.batoi.com/opendelight/docs/controller

## Event

Event happens when an HTTP request is sent to the server-side, and thus an instance of application is created. In Opendelight framework, an Event is identified by *Event ID*, and this is available as `$APP->ID` within the application. For more details on *Event*, please go to https://www.batoi.com/opendelight/docs/event

## Event ID

An **Event** is uniquely identified by *Event ID* and this is the same as `$APP->ID`. For more details on *Event ID*, please go to https://www.batoi.com/opendelight/docs/event

## IDE

Opendelight comes with a tool that developers can use to create, modify and manage their application. This provides an Integrated Development Environment for the developer. IDE is in an early stage currently, and future development will bring in object modeling and remote application deployment into its purview. For more details on IDE, please go to https://www.batoi.com/opendelight/docs/ide

## Lifestream

Opendelight has an inbuilt capability to track the access to the application created through it. This is enabled by default during installation of the framework. You can see the access log of the application in the section meant for Lifestream on Opendelight IDE. For more details, please go to https://www.batoi.com/opendelight/docs/ide

## Model

Model refers to business logic part of the application. This consists of **application classes** and **script includes**. All files of Model are stored in `/model` directory. For more details, please go to https://www.batoi.com/opendelight/docs/model

## Opendelight Array

Opendelight Array, `$_DELIGHT`, is defined by the application instance, and will not be used visually by the developer anywhere. This is an array the framework uses internally. For more details, please go to https://www.batoi.com/opendelight/docs/objects

## Opendelight Database Tables

There are seven database tables created during the installation of Opendelight framework. These tables start with `*od_`, where * is the table prefix string defined during the installation of framework. For more details on Opendelight Database Tables, please go to https://www.batoi.com/opendelight/docs/database

## Opendelight Libraries

Opendelight Library consists of a set of classes that additional capabilities to develop your application. Opendelight Library Classes are located in `/lib` directory. For more details, please go to https://www.batoi.com/opendelight/docs/objects

## Opendelight Objects

The framework provides three core objects to be used during application development - `$DB`, `$APP` and `$USER`. These objects are automatically instantiated and are available in the application. The `$DB` object is responsible for database access and data manipulation. The second object $APP is the instance of application when an HTTP request is made, and carries all related data. The `$USER` object carries data of the user accessing the system (uses *Row Gateway Data Pattern*), and is only instantiated if the controller is *private*. For more details on Delight Objects, please go to https://www.batoi.com/opendelight/docs/objects

## Profile

Opendelight provides an in-built mechanism to manage user's primary details like username, password, email, name and roles as they are critical to handle security aspect of the application. However, a developer can create more attributes

and functionalities to manage user accounts as the application scope demands. This can be achieved by creating a `Profile` class (an **Application Class**) and a corresponding database table. More details can be found at https://www.batoi.com/opendelight/docs/users

## Roles

Each user must be assigned a role as it determines the access privilege and the homepage (after signin) of the concerned user. For learning how to manage roles, please go to https://www.batoi.com/opendelight/docs/users

## Script Includes

A Script Include refers to any PHP file that contains business logic of the application and is a part of **Model**. This file is included within the code for an event to execute respective business logic. Script Includes can be created and managed directly through IDE. However, you can also edit these files in your favourite code editor. These files are located in `/model/script` directory. For more details on Script Includes, please go to https://www.batoi.com/opendelight/docs/model

## Users

Users are entities who will access the application after it is created. Opendelight creates the `admin` user (with access privilege to all aspects of the application and to IDE too) automatically after the framework is installed. Other users can be created through IDE or you can develop your own functionality for this purpose through custom class called `Profile`. For learning how to manage users, please go to https://www.batoi.com/opendelight/docs/users

## View

View refers to part of the application that is responsible for UI. This consists of **View Page Parts**, CSS files, JS files, and also graphics and multimedia assets of the application. Code for **View Page Parts**, CSS files and JS files can be created and managed directly through IDE. However, you can also edit these files in your favourite code editor. For more details, please go to https://www.batoi.com/opendelight/docs/model

## View Page Parts (VPP)

View Page Parts (VPP) are files which contain UI content consisting of HTML and PHP print statements or simple loops

and conditions. They are called Page parts as one or more such files together will create a complete UI that goes to client-side for rendering. These are defined on IDE while creating or editing events, and are available as `$APP->VIEWPARTS` within the application. The content of VPP can be created and managed directly through IDE. However, you can also edit these files in your favourite code editor too. For more details, please go to https://www.batoi.com/opendelight/docs/model

## View Page Variables (VPV)

View Page Variables (VPV) are required to define UI pages. This includes page title (`$APP->PAGEVARS[TITLE]`), page h1 texts (`$APP->PAGEVARS[HEADERTEXT]`), page breadcrumb (`$APP->PAGEVARS[BREADCRUMB]`), etc. While the specified three are automatically created when you install Opendelight framework, any other (as required in the scope of your application) can be created on IDE and be used within the application. For more details, please go to https://www.batoi.com/opendelight/docs/model

# Opendelight Objects

The framework provides three core objects to be used in application development - $DB, $APP and $USER. These objects are automatically instantiated, and are available in the application. The $DB object is responsible for database access and data manipulation. The second object $APP is the instance of application when an HTTP request is made, and carries all related data. The $USER object carries data of the user accessing the system (uses *Row Gateway Data Pattern*), and is only instantiated if the controller is *private*. The interaction of $APP and $USER objects is based on *RBAC* (*Role based Access Control*) scheme as described in Overview of Framework Architecture.

NOTE:
The speciality of Delight objects is that one does not need to instantiate these three objects. These objects are automatically available within the application main scope. You can also make them available within any Model class by accepting them as global within class constructor; i.e.,

```
global $DB, $APP, $USER;
$this->DB   = $DB;
$this->APP  = $APP;
$this->USER = $USER;
```

## Database Object ($DB)

The Database object requires the values of Opendelight Array, `$_DELIGHT`, (usually stored in the file `sys.inc.php` created during the installation of the framework) to get instantiated. The $DB object is basically PDO (PHP Database Object), and can be used in application instance to perform database operations. As PDO supports many major databases including MySQL, PGSQL, and Oracle, etc., developer can choose database freely based on situation and requirements. It is required that all database queries must be executed through prepared statements – well, that gives immunity against SQL injection.

## Application Object ($APP)

The application object requires application event ID (that is available on `$_REQUEST`) to get instantiated. It depends other critical parameters like `ID` (event ID that determines the application instance) or `IDN` (This is name of application

instance (availed from $_REQUEST super global and can be used instead of ID). The later is usually employed for beautifying URL, and also for tagging event's ID for readability during development. After instantiation of $APP object, the following values becomes available to the application instance:

| Attributes | Description |
|---|---|
| ID | This is the instance ID of Application or Event ID |
| EVENTVERIFIER | This is an expression that will be validated before the application instance is executed. |
| FORMRULES | This is an array which store sanitization information for individual form fields values received through $_REQUEST. If there is no such requirement for any application instance, this array remains empty. |
| VIEWPARTS | This is an array that stores list of view parts (refer to sub-section on View Parts in this section) in the order of display. |
| PAGEVARS | This is an array that stores strings alongside keys:<br>TITLE: Title of Page<br>H1: Heading of Page<br>H2: Subheading |
| BASEURL | Base URL of the application |
| URL | Full URL of the application instance |
| CONFIG | Stores user-defined configuration settings |
| SYS | [NAME]: Name of Application<br>[AUTHOR]: Description of Application including license information, links, etc.<br>[DESCRIPTION]: The author of application |
| TABLEPREFIX | This stores the table prefix of user-defined tables |

## User Object ($USER)

The User object requires $APP object to get instantiated, but you do not need to be worried – it is taken care of automatically. On the other hand, $USER is instantiated only if $APP->ISPUBLIC returns false (managed within the controller – you do not need to manage this code, rather will mark the controller as *private* on IDE). The $USER object depends on critical parameters like User's ID and VERIFIER (the verifier is a unique string generated at the time of user

creation and differentiates users having common session pool in the server for different applications – again you do not need to be worried about this as these things are taken care of automatically), but avails it from `$_SESSION` super global. After instantiation of `$USER` object, the following values becomes available to the application instance:

| Attributes | Description |
|---|---|
| ID | This is the `ID` of user accessing the application. There will be no `ID` if the controller is *public*; i.e., `$APP->ISPUBLIC` returns true. |
| USERNAME | This is username |
| EMAIL | This is user email |
| FULLNAME | This is fullname of user |
| LASTLOGIN | This is datetime information of user when he/she signed last |

# Opendelight Array

These are system variables and MUST NOT be used during the application development. The table below lists all elements of Opendelight Array:

| Variable Name | Location | Description |
|---|---|---|
| $_DELIGHT[DSN] | System Config file | Database Source Name |
| $_DELIGHT[DBUSER] | System Config file | Database Username |
| $_DELIGHT[DBPASSWORD] | System Config file | Database Password |
| $_DELIGHT[TABLEPREFIX] | System Config file | Database Table Prefix |
| $_DELIGHT[CTRID] | Controller File | Controller ID |

When Opendelight is installed, the first four elements are defined. These values are stored in the file `sys.inc.php`, and are the same for all instances of the application. The fifth element is defined during runtime when a controller receives HTTP request.

NOTE:

Apart from the Opendelight Array, a few elements from `$_REQUEST/ $_SESSION` Super Globals should not be used as they store system runtime data. These are listed in the table below:

| Variable Name | Location | Description |
|---|---|---|
| $_SESSION[USERID] | Session Super Global | User ID |
| $_SESSION[IDVERIFIER] | Session Super Global | User ID Verifier – additional verifyer is a string generated randomly and is unique to the user across the server. This is used to distinguish user from |

| | | |
|---|---|---|
| | | others if more than one application with same user object data share the same session pool in the server or cloud. |
| $_SESSION[REQUESTURL] | Session Super Global | This is the URL user requests which gets stored in session and is used to come back to the same location after signing in. |
| $_REQUEST[ID] | Request Super Global | This stores the event ID of application instance |
| $_REQUEST[IDN] | Request Super Global | This is name of application instance. This is used for beautifying URL, and also for tagging event's ID for readability during development |

# Opendelight Libraries

Opendelight Libraries are classes developed or distributed as the part of Opendelight framework to facilitate additional objects to develop your application. These classes are available in `/lib` directory, and are autoloaded when instantiated (i.e., you do not need to include these class files in the application when you want to instantiate them). The names of these classes usually start with `Delight_`.

The following are the classes currently available as Opendelight Libraries:

| Attributes | Description |
|---|---|
| Delight_Grid | This class is used to perform backend operations for jqgrid plugin of jQuery (that is used for grid display on application). You need to include jqgrid script files (http://www.trirand.com/blog/jqgrid/jqgrid.html) to use this class. |
| Delight_Form | This class is used for processing form for sanitizing form input data and for performing additional validations. This uses other opensource scripts, but have been packaged inside the distribution. `Sanitize` is a part of this class. |
| Delight_Captcha | This class is used to display and verify CAPTCHA in public forms |
| Delight_Sign | This class is used for performing signing in, signing out and retrieving password in the application |

# Application Database Structure

When we install Opendelight, seven database tables are created with name as *od_* as below. These tables store application information and are critical to application created with Opendelight.

| Table Name | Description | Fields along with attributes and explanation |
|---|---|---|
| *od_controller | Stores module information | 1. ctrid – INT(11)<br>2. ctrname – VARCHAR(255)<br>3. ispublic – ENUM('0','1') [0=public, 1=private]<br>4. defaulteventid – INT(11)<br>5. sortorder – INT(11)<br>6. ctrstatus – ENUM('0','1') [0=inactive, 1=active]<br>7. `signinctrid – INT(11)` [ctrid of corresponding signin controller if it is a private controller]<br>8. lastupdate - DATETIME |
| *od_event | Stores event information | 1. eventid – INT(11)<br>2. eventname – VARCHAR(255)<br>3. ctrid – INT(11)<br>4. sortorder – INT(11)<br>5. estatus – ENUM('0','1') [0=inactive, 1=active]<br>6. eventverifier – VARCHAR(255)<br>7. formrules – LONGTEXT<br>8. blcode - LONGTEXT<br>9. viewparts – TEXT<br>10. pagevars - TEXT<br>11. roles – VARCHAR(255)<br>12. lastupdate - DATETIME |
| *od_config | Stores user defined application configurations | 1. configid – INT(11)<br>2. configname – VARCHAR(255)<br>3. description – VARCHAR(255) |

| Table Name | Description | Fields along with attributes and explanation |
|---|---|---|
| | | 4. configvalue – VARCHAR(255)<br>5. sortorder – INT(11) |
| *od_pagevars | Stores pagevar array keys | 1. pagevarid – INT(11)<br>2. pagevarkey – VARCHAR(255) |
| *od_user | Stores essential user information | 1. userid – INT(11)<br>2. idverifier – VARCHAR(255)<br>3. username – VARCHAR(255)<br>4. password – VARCHAR(255)<br>5. email – VARCHAR(255)<br>6. firstname – VARCHAR(255)<br>7. lastname – VARCHAR(255)<br>8. userstatus – ENUM('0','1','2') [0=inactive, 1=active,2=archived]<br>9. roleid – INT(11)<br>10. lastlogin – DATETIME |
| *od_role | Stores role information | 1. roleid – INT(11)<br>2. rolename – VARCHAR(255)<br>3. sortorder – INT(11)<br>4. defaultctrid – INT(11)<br>5. defaulteventid – INT(11) |
| *od_sys | Store system information | 1. sysid – INT(11)<br>2. appname – VARCHAR(255)<br>3. author – VARCHAR(255)<br>4. description – TEXT<br>5. `baseurl` – `VARCHAR(255)` [the base URL of the application]<br>6. sysstatus – ENUM('0','1') [0=not in production, 1=in production] |

During the course of development, developers add more tables depending on the scope of application.

# Application File Structure

The opendelight framework reinforces correct development paradigm through different prescriptions and architectural procedures. Having a clear directory and file structure is one of them. This creates separation of business logic from UI design on one hand, and user development from framework's upgrades and IDE on the other. After installation, the framework creates the following directory structure:

| Directories with respect to application root | Description of directories and their usage |
|---|---|
| /delight-ide | This directory contains all files of Opendelight IDE. You can even delete this directory after completing development, and that will not affect the running of your application in any way. |
| /cache | The cache files created by application is stored in this directory. This includes static files created by application beforehand and served to client-side for improving application performance and to reduce load on server. |
| /ext | This directory meant to store external scripts that are to be added to an application being developed by Opendelight framework |
| /files | This directory is the storage of uploaded files to the application. You can save uploaded files in `/<year>/<month>/<date>` form or in any other form as per the convenience of application requirements. |
| /lib | The scripts and classes provided by Opendelight are stored in this directory. Please note that the files in this directory will be updated if you do upgrades of the framework in future with newer versions. |
| /log | If you enable lifestream on IDE, the application will generate log everytime the application is accessed. This log information is stored as text files in this directory. |
| /model | All files of Model or business logic are stored here. The Model classes or application classes are stored in a subdirectory `/class`, and independent scripts are stored in `/script`. |

| | |
|---|---|
| /view | This directory contains all folders and files related to view – client-side code and files. Different types of view files are stored in their own subdirectories like the following:<br>`/css` – CSS files<br>`/font` – Font files<br>`/html` – HTML and XML files<br>`/images` – Image files<br>`/js` – Javascript files<br>`/multimedia` – Audio and Video files<br>`/src` – High resolution images, etc. |

The framework also provided a few key files in the application root that are critical for application running. The following table provides details of each of these files:

| Opendelight files in application root | Description of files and their usage |
|---|---|
| /index.php | This is the default controller of application as anybody will access this file when they go to `<application-base-url>`. In case of public website, this controller can be made *public*. In case of web application, this controller can be made *private*, and `sign.php` be made its *Sign In* controller. |
| /load.delight.php | This is the most important file of Opendelight framework and implements framework's logic within application. This file is automatically included within each controller when created on IDE. |
| /load.view.php | This file is automatically included within each controller and is responsible to call viewparts in their specified order. |
| /sign.php | This is the default *Sign In* controller, and can be used to implement user sign in, forgot password and sign out functionalities of the application. You can customise the View Parts of events of sign.php controller to implement any specific look and feel. |

| | |
|---|---|
| /sys.inc.php | This file stores database settings, and is created during the installation of framework. |

# Controller in Opendelight Framework

A Controller is a PHP file that takes the HTTP request, and manages the execution path of the application (instance of application) at server-side. An application can have more than one controller files when required. A Controller file can define public URLs or private URLs (i.e., password protected URLs) as the situations demand. However, a Controller file cannot have both public and private URLs, and you need separate Controller files for this.

A Controller file is defined and edited by the Opendelight IDE, and requires the developer to specify the controller file name. The Controller file name can be preppended by path from root of application folder if it is located inside an inner directory, but never include any slash in the beginning. For example,

- Controller name is `mycontroller.php` if it is located in application root directory.
- Controller name is `test/mycontroller.php` if it is located inside directory `test` located in application root directory.

NOTE:
We have decided to use the name as mycontroller.php in the above example, but you can name anything. Please note further that you can also choose other file extension for your controller file if you are ready to implement that as a PHP script through your apache configuration or .htaccess)

Please note that you already get two controllers when you install the framework – `index.php` and `sign.php`:

- `index.php` is the default controller of application as anybody will access this file when they go to `<application-base-url>`. In case of public website, this controller can be made *public*. In case of web application, this controller can be made *private*, and `sign.php` be made its *Sign In* controller.
- `sign.php` is the default *Sign In* controller, and can be used to implement user sign in, forgot password and sign out functionalities of the application. You can customise the View Parts of events of sign.php controller to implement any specific look and feel.

What Does A Controller Do in Opendelight?

- Declares special PHP environment setting, if any (these are particularly `php.ini` directive(s))
- Starts Session with the line `session_start();`
- Defines `$_CTRID`. This will be the ctrid field value at `*od_controller` table for corresponding controller file.
- Includes the file `load.delight.php`. This act will instantiate three Delight objects: `$DB`, `$APP` and `$USER` (if it is *private* controller)
- Calls BL based on application instance ($APP->ID), and loads data for like `$APP->VIEWPARTS`, `$APP->$FORMRULES` and `$APP->$PAGEVARS`
- Finally, calls View Page Parts as defined for the application instance (in `$APP->VIEWPARTS`). If no View Page Part is defined for a particular instance, data arrays as obtained from BL calls, are sent back to UI by the Controller (usual in case of AJAX calls from UI).

# Event in Opendelight Framework

Event refers to an application instance. When a distinct HTTP request is made to application, it is accompanied by an value of *Event ID* or *Event Name*. In Opendelight framework, an Event is uniquely identified by *Event ID* (available as `$APP->ID` within the application), or by Event Name (available as `$APP->IDN` within the application). The later is usually employed for beautifying URL, and also for tagging event's `ID` for readability during development.

An event can be defined for a Controller on IDE. You can add an event with the following information:

1. **Event Name, `$APP->IDN`:** This is any string of alphanumeric characters without any white space or special characters, but can have dashes or underscores. *Event Name* must be unique for an application. *Event ID* is automatically assigned to an event upon addition.
2. **Set as Default:** You can specify if this event is the default event; i.e., this event is invoked when the controller is called without specifying the *Event ID* or *Event Name*. You should have only one event to be set as default within a Controller.
3. **Roles:** You can specify the user roles that can access this event of the application. You can specify multiple roles. The role `Administrator` is selected by default and cannot be modified.
4. **Event Verifier:** Additional boolean expression that needs to be satisfied to make the event executed successfully; e.g., `$iMyVariable == 'N'`
5. **Form Rules:** This is an array with information about form validation and sanitization as required in the scope of the application. This array is used with `Delight_Form` class (an Opendelight Library Class) to validate and sanitize form data at the server side. For example, if we have two text-box values like *Username* and *Password* that we want to validate for and sanitise the input, we write the array as

```
"txtUsername" => array("validate" => 1,
"validation_type" => "required",
"reg_exp" => "", "error_message" => "This
field is required", "sanitize" => 1,
"sanitize_type" => "safe"),"txtPassword"
```

```
=> array("validate" => 1,
"validation_type" => "required",
"reg_exp" => "", "error_message" => "This
field is required", "sanitize" => 1,
"sanitize_type" => "safe")
```

and use the following code within the Event Case (next field with code editor on the same page on IDE):

```
$oForm = new Delight_Form($_POST, $APP-
>FORMRULES);
$aForm = iterator_to_array($oForm);
```

Now, the variable `$aForm` carries validated and sanitized posted form data, which can be used within the application.

6. **Code for Calling Application Objects and Including External Scripts:** Here, you instantiate Application classes or include Script Includes to perform business logic operations.
7. **View Page Parts:** A comma-separated ordered list of View Page Part files to be called for creating UI
8. **View (Web) Page Variables:** `TITLE`, `HEADERTEXT`, BREADCRUMB are defined here along with any other View Page Variables that are defined in sections specified.

An event can be made active or inactive on IDE. The list of the events can be ordered for display (used for readability purpose on IDE only).

# Model in Opendelight Framework

Model refers to business logic part of the application. This consists of *application classes* and *script includes*. All files of Model are stored in `/model` directory.

An **Application Class** refers to any PHP class that contains business logic of the application and is a part of Model. Application classes can be created and managed directly through IDE. However, you can also edit these classes in your favourite code editor. These classes are located in `/model/class` directory.

NOTE:

- The file name of an Application Class should be exactly the same as class name but should have extension as `.cls.php`. There should be no white space in the class name or in corresponding filename.
- Do not create name of Application Class equal to Open delight library class which always starts with `Delight_`
- There is no need to include Application Class files as the classes are autoloaded.

A **Script Include** refers to any PHP file that contains business logic of the application and is a part of Model. This file is included in an event to execute respective business logic. Script Includes can be created and managed directly through IDE. However, you can also edit these files in your favourite code editor. These files are located in `/model/script` directory. The file name of a Script Include should have file extension as `.inc.php`.

The other thing that you can manage about Model is the **Configuration Variables** of application. While developing an application, we tend to use a number of magic numbers. Ideally we store these in a separate file, and in turn, include it everytime within the application. With Opendelight, we do not need to do any such effort; rather we create these as configuration variables on IDE, and then use them as a property (date type is associative array) of `$APP` object, `$APP->CONFIG`, within the application. You can add a Configuration Variable by filling *Variable Name, Description* and *Value of Variable* on IDE. You can arrange the display order of the

created variables; however, this is just for display purpose on IDE (for readability of list of variables for developer), and do not have any other requirement during application development.

# View in Opendelight Framework

View refers to part of the application that is responsible for UI. This consists of **View Page Parts**, CSS files, JS files, and also graphics and multimedia assets of the application. Code for **View Page Parts**, CSS files and JS files can be created and managed directly through IDE. However, you can also edit these files in your favourite code editor too.

**View Page Parts (VPP)** are files which contain UI content consisting of HTML and PHP print statements or simple loops and conditions. They are called Page parts as one or more such file will create a complete UI that goes to client-side for rendering. These are defined on IDE while creating or editing events and are available as `$APP->VIEWPARTS` within the application. The content of VPP can be created and managed directly through IDE. However, you can also edit these files in your favourite code editor. For example, there are three VPPs which make UI in the default event of `index.php` Controller (when Opendelight is installed, these are created to start with): `'header-main.tpl.php', 'hello-world.tpl.php', 'footer-main.tpl.php'`
View page Parts are connected to an event on the event definition page on IDE. In many cases, VPPs are optional, particularly in AJAX requests from UI. In such cases, the data arrays received from Model are returned directly without calling view. Sometimes, one or more VPPs are required even in AJAX calls.

**View Page Variables (VPV)** are required to define UI pages. This includes page title (`$APP->PAGEVARS[TITLE]`), page h1 texts (`$APP->PAGEVARS[HEADERTEXT]`), page breadcrumb (`$APP->PAGEVARS[BREADCRUMB]`), etc. While the specified three are automatically created when you install Opendelight framework, any other (as required in the scope of your application) can be created on IDE and be used within the application. You can also include PHP variables within the VPV definition if required.

# Users and Roles

Users are entities who will access the application after it is created. Opendelight creates the `admin` user (with access privilege to all aspects of the application and to IDE too) automatically after the framework is installed. Other users can be created through IDE or you can develop your own functionaly for this purpose through custom class called `Profile`.

NOTE:
Opendelight provides an in-built mechanism to manage user's primary details like username, password, email, name and roles as they are critical to handle security aspect of the application. However, a developer can create more attributes and functionalities to manage user accounts as the application scope demands. This can be achieved by creating a `Profile` class (an **Application Class**) and a corresponding database table.

The username `admin` (case-sensitive) is the super user which can access Opendelight IDE and can develop, manage and access entire application. This user gets such a right through a default role assigned to it – `Administrator`. Neither this username nor its association with the role `Administrator` can be modified.

Each user must be assigned a **role** as it determines the access privilege and the homepage (after signin in) of the concerned user. When you install the framework, the default role is `Administrator`, and it is assigned automatically to the user `admin`. Neither this role nor its association with the user `admin` can be modified.

# Settings of Application with Opendelight Framework

**Application settings** refer to setup information that are used while installing the framework, thus setting up the application with Opendelight framework. This includes *application name, description, author name, email address of admin user, URL of application* and *database settings*. Database settings include *DSN, Database Name, DB user* and *Password*, and *Table Prefix* (the string that will be preppended to each table of the application including **Opendelight Database Tables**).

You can edit the application settings on IDE. But you need to be careful about Application Base URL as it may make the application disfunctional. However, you can manage your Lifestream log (enabling and disabling) here apart from other aspects of application. There is no facility to edit database settings for an application as there is a separate utility (currently under development) to help you migrate your application to different database and different location.

# Opendelight IDE

Opendelight IDE (Integrated Development Environment) is a web based tool to develop and manage your application with Opendelight. This tool is included along with the Opendelight download on its Download page.

IDE can be accessed at the URL: <the-base-url-of-application>/delight-ide

IDE provides utilities to

- view recently edited events and controllers apart from blog posts about Opendelight on ADII website and links to framework documentation (in fact, each screen on Opendelight IDE provides link to documentation contextually)
- manage controllers and events with each
- manage Model (Application Classes, Script Includes, and Configuration Variables)
- manage View (View Page Parts, View Page Variables, JavaScript Files, and CSS Files)
- manage users and their roles
- manage Lifestream
- view and edit pplication Settings
- change admin password

IDE is in an early stage currently, and future development will bring in object modeling and remote application deployment into its purview.

# Application Life-stream

Opendelight has an inbuilt capability to track the access to the application created through it. This is enabled by default during installation of the framework. You can see the access log of the application in the section meant for Lifestream on Opendelight IDE.

Each day log is stored in separate text files, and creates a new row on the Lifestream page on IDE. The log files are stored in the directory `/log`, and rows of log file looks like the following (two lines of a successfully executed event have been presented for illustration):

25 March 2010, 7:10 am - Application accessed by user admin through URL /appl/index.php?ID=6 from IP 127.0.0.1.
25 March 2010, 7:10 am - Application executed in 0.1765148639679 sec.

The first line above describes the access by user (if it is public access, no user information will be present), and the second line describes the successful execution of the event. In case of failure in execution of event, the second line will be absent in the log. As the log stores critical information like user, URL, IP from which access is done and time for execution of event, it is very useful to study the performance of application.

The IDE has utility to delete log of any particular day, and also to remove the entire history of access log.

NOTE:

You may like to disable the Lifestream log (this can be done by editing the Application Settings) as it produces daily log files and thus consumes disk space. But it is advisable to keep it enabled, and to remove the history periodically. The removal of history is extremely critical on high-traffic sites or applications.

# Coding Standards and Conventions

The opendelight framework reinforces correct development paradigm through IDE and a set of prescribed guidelines for development. Coding standards and conventions do help in this direction. Please find the following list of guidelines as recommendations from us, but you are free to choose you style : )

## Server-side Development with PHP

1. Keep one class one file decorum so that we can use php-doc for technical documentation. More information about PHPDoc can be found here: http://www.phpdoc.de/
2. Always use <?php ?> to delimit PHP code, not the <? ?> shorthand.
3. Use an indent of 4 spaces, with no tabs.
4. Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.
5. Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. There should be one space on either side of an equal sign used to assign the return value of a function to a variable.
6. Function declarations follow the "one true brace" convention. Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate.
7. Inline documentation for classes should follow the PHPDoc convention, similar to Javadoc.
8. Non-documentation comments are strongly encouraged. A general rule of thumb is that if you look at a section of code and think "Wow, I don't want to try and describe that", you need to comment it before you forget how it works.
9. C style comments (/* */) and standard C++ comments (//) are both fine. No Perl/shell style comments (#) should be used. Comments may be added at the start of the program file, before starting functions, or classes or variables, block of code.

10. Use "example.com", "example.org" and "example.net" for all example URLs and email addresses, per RFC 2606.
11. Identifier names should be explanatory, and should be in word-case. Also the first letter of the variable or parameter or IDs must tell the nature thereof. Examples (please note carefully) are:
    a. Class: ClassName
    b. Object: oObjectName
    c. Function or method Name: `functionName()`
    d. Constants: CONSTANTNAME
    e. String Variable: `$sVariableName`
    f. Array Variable: `$aVariableName`
    g. Integer Variable: `$iVariableName`
    h. Float Variable: `$fVariableName`
    i. Boolean Variable: `$bVariableName`

## Client-side Development with HTML and CSS

1. Style-sheet must be used for all properties of the HTML elements.
2. HTML code must be XHTML1.0 strict: http://validator.w3.org/
3. CSS code must comply with CSS2 standard: http://jigsaw.w3.org/css-validator/
4. C style comments (`/* */`) and standard C++ comments (`//`) are both fine.
5. ID or NAME attributes should be explanatory, and should be in word-case. Examples (please note carefully) are:
    a. CSS Style Class: `ClassName`
    b. HTML ID Attribute: `attributeName`
    c. Form ID: `formId`
    d. Text Box Name/ID: `txtFieldName`
    e. Text Area Name/ID: `taFieldName`
    f. Drop Menu Name/ID: `dmFieldName`
    g. Radio Button Name/ID: `rbFieldName`
    h. Checkbox Name/ID: `chkFieldName`
    i. Button Name/ID: `btnFieldName`
    j. Hidden Field Name: `hidFieldName`

## Client-side Development with Javascript

1. C style comments (`/* */`) and standard C++ comments (`//`) are both fine.

2. Identifier names should be explanatory, and should be in word-case. Also the first letter of the variable or parameter or IDs must tell the nature thereof. Examples (please note carefully) are:
    a. Object: oObjectName
    b. Method Name: `functionName()`
    c. String Variable: `sVariableName`
    d. Array Variable: `aVariableName`
    e. Integer Variable: `iVariableName`
    f. Float Variable: `fVariableName`
    g. Boolean Variable: `bVariableName`

# Installing Opendelight Framework

Installing Opendelight is quite easy and fast. Please follow the following steps to install Opendelight:

1. Download Opendelight ZIP from the Download page and unzip into any folder or subfolder in your web root ("web root" is the parent folder or directory in your PC or server which can be accessed through "http").
2. Go to the URL http://your-domain/folder1/folder2/.../folderN/opendelight-folder/delight-ide/install.php
3. Click "Setup Application", and you will be taken to a page to fill the details of your application. Fill the requisite details and click "submit". You are done with.

NOTE:

- Check your application URL (where you are going to develop your new application) is correct. If you find any disparity, please rectify at this point. Make sure that the URL does not have a '/' at the end.
- The email you enter during the setup will be used as the email ID of "admin" user and be used for password resetting apart from any application-related tasks.
- Currently, Opendelight prescribes use of PHP Data Object (PDO) as the data-access abstration layer and prepared statements for querying databases. Some examples of DSN for different supported databases (by PDO and consequently by Opendelight) are:
  **MYSQL:**
  `mysql:host=yourhostname;port=portno;dbname=yourdbname`
  **PostgreSQL:**
  `pgsql:host=yourhostname;port=portno;dbname=yourdbname`
  **Oracle:**
  `oci:host=yourhostname;port=portno;dbname=yourdbname`
  **SQLite:** `sqlite:pathname/dbname.db`
- Table-prefix will be a string that will be preppended to each table in the database. In case of Opendelight system tables, they will be automatically be done. In case of the new tables you create, make sure that you do the same.
- You can remove the `/delight-ide` folder on your live site if you wish – this will not affect the working of your live site in any way. You can always upload this folder at any later stage if you would like to modify your application.

# Upgrading Opendelight Framework

Upgrading Opendelight is quite easy and fast. Please follow the following steps to install Opendelight:

1. Download Opendelight ZIP from the Download page and copy the following to your application root overwrite the respective files/folders:
    a. `/delight-ide` and all files inside it
    b. `/lib` and all files inside it
    c. /load.delight.php
    d. /load.view.php
2. Go to the URL `http://your-application-url/delight-ide/upgrade.php`, and follow the instructions there. You will be done with upgrade.

# Future Roadmap of Opendelight

We have ambitious plan to enhance the framework further to make the Rich Internet Application development faster and smoother. The following are the plan of action in the coming months:

1. Support for mobile application development
2. Support for offline application development on desktop and mobile
3. Native database management tool
4. *Opendelight Libraries* for web services, emailing, data record handling and other routine tasks
5. Apart from the current access control (RBAC) scheme, a record based access scheme has been prescribed for applications that need more granual access of objects by users. We are planning to support this natively.

Write to us if you wish more – labs@batoi.com

Check out our Labs page for latest updates about our different projects – https://www.batoi.com/labs